Python and R Project Code:

Ritai Na

1. Introduction:

Shown below is a brief Introduction to Wind Turbine Status Analysis Project in the course ***Machine Learning and Big Data***:

You may check the code in Github link Below.

Github Project Link: https://github.com/BlueFamous/Machine-Learning-Course-Project

2. Data Preprocessing:

(1) Dimension Reduction:

The original dataset consist of over 67 attributes and the total lines of data is over 1 million. To increase efficiency, I applied Lasso to make dimension regression using R.

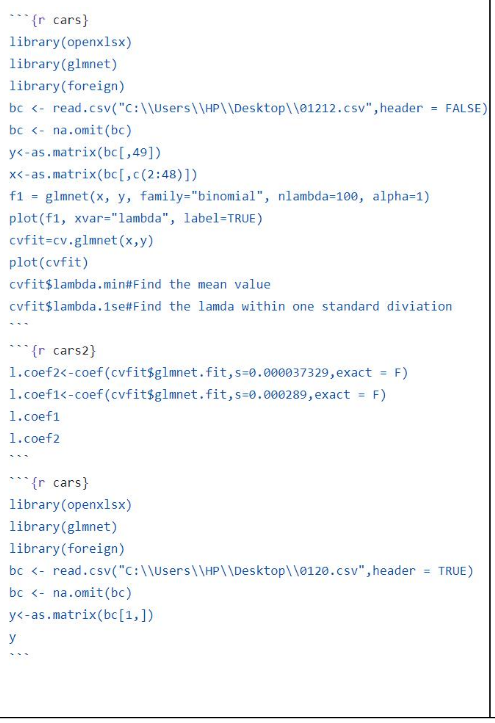
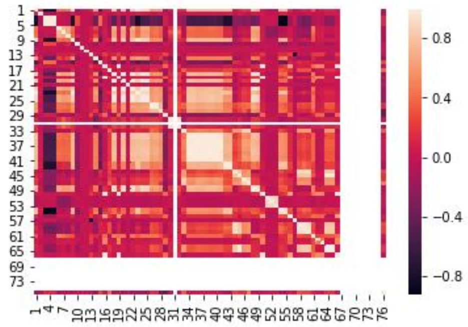
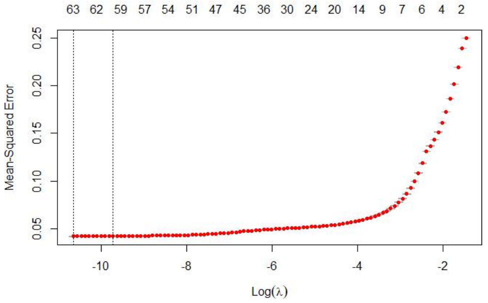
(2) Oversampling to settle sample imbalance:

The original dataset is imbalanced in labels. Hence, I applied Borderline-SMOTE to settle this issue. The code and the result is shown below.

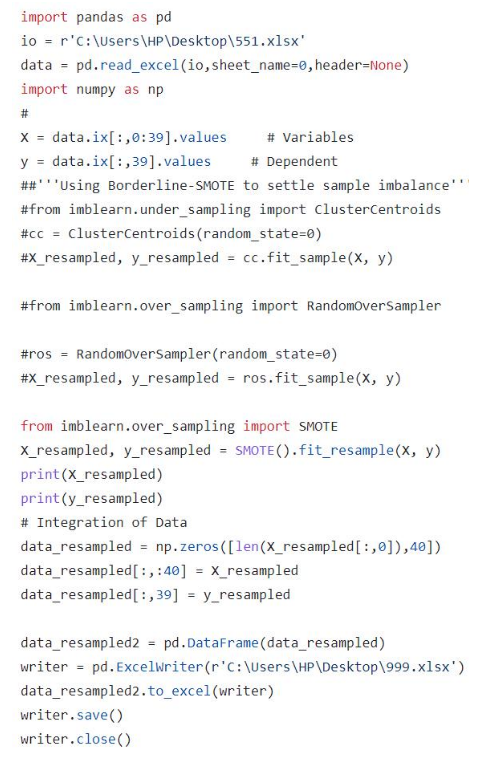
(3) Two-stage Transfer Adaboost:

Applied two-stage transfer adaboost to predict the labels.

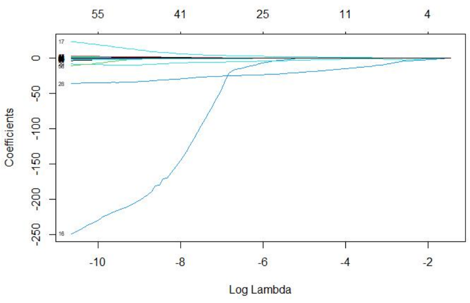
Step	Original Code	Result

Dimension Reduction	
	```
	{r cars}
	library(openxlsx)
	library(glmnet)
	library(foreign)
	bc <- read.csv("C:\\Users\\HP\\Desktop\\01212.csv",header = FALSE)
	bc <- na.omit(bc)
	y<-as.matrix(bc[,49])
	x<-as.matrix(bc[,c(2:48)])
	f1 = glmnet(x, y, family="binomial", nlambda=100, alpha=1)
	plot(f1, xvar="lambda", label=TRUE)
	cvfit=cv.glmnet(x,y)
	plot(cvfit)
	cvfit$lambda.min#Find the mean value
	cvfit$lambda.1se#Find the lamda within one standard diviation
	```
	```
	{r cars2}
	l.coef2<-coef(cvfit$glmnet.fit,s=0.000037329,exact = F)
	l.coef1<-coef(cvfit$glmnet.fit,s=0.000289,exact = F)
	l.coef1
	l.coef2
	```
	```
	{r cars}
	library(openxlsx)
	library(glmnet)
	library(foreign)
	bc <- read.csv("C:\\Users\\HP\\Desktop\\0120.csv",header = TRUE)
	bc <- na.omit(bc)
	y<-as.matrix(bc[1,])
	y
	```
		
		
		

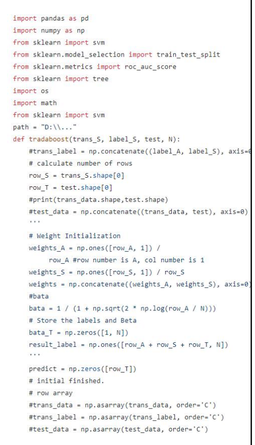
Oversampling



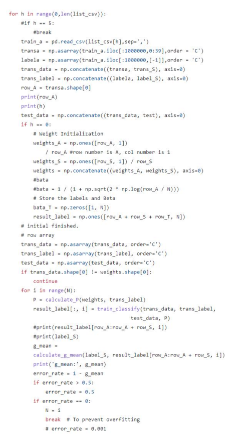
```
import pandas as pd
io = r'C:\Users\HP\Desktop\551.xlsx'
data = pd.read_excel(io,sheet_name=0,header=None)
import numpy as np
#
X = data.ix[:,0:39].values     # Variables
y = data.ix[:,39].values     # Dependent
##'''Using Borderline-SMOTE to settle sample imbalance''
#from imblearn.under_sampling import ClusterCentroids
#cc = ClusterCentroids(random_state=0)
#X_resampled, y_resampled = cc.fit_sample(X, y)

#from imblearn.over_sampling import RandomOverSampler

#ros = RandomOverSampler(random_state=0)
#X_resampled, y_resampled = ros.fit_sample(X, y)

from imblearn.over_sampling import SMOTE
X_resampled, y_resampled = SMOTE().fit_resample(X, y)
print(X_resampled)
print(y_resampled)
# Integration of Data
data_resampled = np.zeros([len(X_resampled[:,0]),40])
data_resampled[:,:40] = X_resampled
data_resampled[:,39] = y_resampled

data_resampled2 = pd.DataFrame(data_resampled)
writer = pd.ExcelWriter(r'C:\Users\HP\Desktop\999.xlsx')
data_resampled2.to_excel(writer)
writer.save()
writer.close()
```



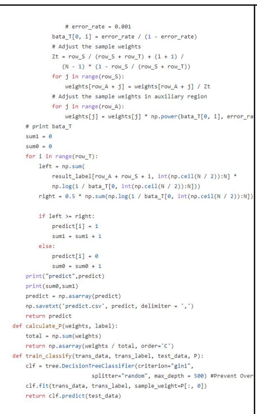
Two-stage Tradaboost Code



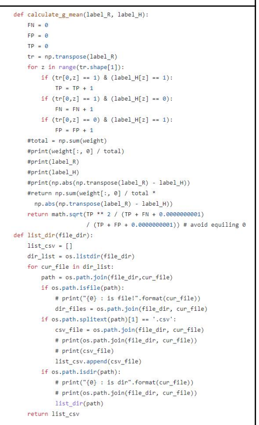
```
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn import tree
import os
import math
from sklearn import svm
path = "D:\\..."
def tradaboost(trans_S, label_S, test, N):
    #trans_label = np.concatenate((label_A, label_S), axis=0
    # calculate number of rows
    row_S = trans_S.shape[0]
    row_T = test.shape[0]
    #print(trans_data.shape,test.shape)
    #test_data = np.concatenate((trans_data, test), axis=0)
    ...

    # Weight Initialization
    weights_A = np.ones([row_A, 1]) /
         row_A #row number is A, col number is 1
    weights_S = np.ones([row_S, 1]) / row_S
    weights = np.concatenate((weights_A, weights_S), axis=0)
    #bata
    bata = 1 / (1 + np.sqrt(2 * np.log(row_A / N)))
    # Store the labels and Beta
    bata_T = np.zeros([1, N])
    result_label = np.ones([row_A + row_S + row_T, N])
    ...

    predict = np.zeros([row_T])
    # initial finished.
    # row array
    #trans_data = np.asarray(trans_data, order='C')
    #trans_label = np.asarray(trans_label, order='C')
    #test_data = np.asarray(test_data, order='C')
```



```
for h in range(0,len(list_csv)):
    #if h == 5:
        #break
    train_a = pd.read_csv(list_csv[h],sep=',')
    transa = np.asarray(train_a.iloc[:1000000,0:39],order = 'C')
    labela = np.asarray(train_a.iloc[:1000000,[-1]],order = 'C')
    trans_data = np.concatenate((transa, trans_S), axis=0)
    trans_label = np.concatenate((labela, label_S), axis=0)
    row_A = transa.shape[0]
    print(row_A)
    print(h)
    test_data = np.concatenate((trans_data, test), axis=0)
    if h == 0:
        # Weight Initialization
        weights_A = np.ones([row_A, 1])
            / row_A #row number is A, col number is 1
        weights_S = np.ones([row_S, 1]) / row_S
        weights = np.concatenate((weights_A, weights_S), axis=0)
        #bata
        #bata = 1 / (1 + np.sqrt(2 * np.log(row_A / N)))
        # Store the labels and Beta
        bata_T = np.zeros([1, N])
        result_label = np.ones([row_A + row_S + row_T, N])
    # initial finished.
    # row array
    trans_data = np.asarray(trans_data, order='C')
    trans_label = np.asarray(trans_label, order='C')
    test_data = np.asarray(test_data, order='C')
    if trans_data.shape[0] != weights.shape[0]:
        continue
    for i in range(N):
        P = calculate_P(weights, trans_label)
        result_label[:, i] = train_classify(trans_data, trans_label,
                                            test_data, P)
        #print(result_label[row_A:row_A + row_S, i])
        #print(label_S)
        g_mean =
        calculate_g_mean(label_S, result_label[row_A:row_A + row_S, i])
        print('g_mean:', g_mean)
        error_rate = 1 - g_mean
        if error_rate > 0.5:
            error_rate = 0.5
        if error_rate == 0:
            N = i
            break  # To prevent overfitting
            # error_rate = 0.001
```



```
            # error_rate = 0.001
        bata_T[0, i] = error_rate / (1 - error_rate)
        # Adjust the sample weights
        Zt = row_S / (row_S + row_T) + (i + 1) /
          (N - 1) * (1 - row_S / (row_S + row_T))
        for j in range(row_S):
            weights[row_A + j] = weights[row_A + j] / Zt
        # Adjust the sample weights in auxiliary region
        for j in range(row_A):
            weights[j] = weights[j] * np.power(bata_T[0, i], error_ra
    # print bata_T
    sum1 = 0
    sum0 = 0
    for i in range(row_T):
        left = np.sum(
            result_label[row_A + row_S + i, int(np.ceil(N / 2)):N] *
            np.log(1 / bata_T[0, int(np.ceil(N / 2)):N]))
        right = 0.5 * np.sum(np.log(1 / bata_T[0, int(np.ceil(N / 2)):N])

        if left >= right:
            predict[i] = 1
            sum1 = sum1 + 1
        else:
            predict[i] = 0
            sum0 = sum0 + 1
    print("predict",predict)
    print(sum0,sum1)
    predict = np.asarray(predict)
    np.savetxt('predict.csv', predict, delimiter = ',')
    return predict
def calculate_P(weights, label):
    total = np.sum(weights)
    return np.asarray(weights / total, order='C')
def train_classify(trans_data, trans_label, test_data, P):
    clf = tree.DecisionTreeClassifier(criterion="gini",
                        splitter="random", max_depth = 500) #Prevent Over
    clf.fit(trans_data, trans_label, sample_weight=P[:, 0])
    return clf.predict(test_data)
```



```
def calculate_g_mean(label_R, label_H):
    FN = 0
    FP = 0
    TP = 0
    tr = np.transpose(label_R)
    for z in range(tr.shape[1]):
        if (tr[0,z] == 1) & (label_H[z] == 1):
            TP = TP + 1
        if (tr[0,z] == 1) & (label_H[z] == 0):
            FN = FN + 1
        if (tr[0,z] == 0) & (label_H[z] == 1):
            FP = FP + 1
    #total = np.sum(weight)
    #print(weight[:, 0] / total)
    #print(label_R)
    #print(label_H)
    #print(np.abs(np.transpose(label_R) - label_H))
    #return np.sum(weight[:, 0] / total *
      np.abs(np.transpose(label_R) - label_H))
    return math.sqrt(TP ** 2 / (TP + FN + 0.0000000001)
                    / (TP + FP + 0.0000000001)) # avoid equiling 0
def list_dir(file_dir):
    list_csv = []
    dir_list = os.listdir(file_dir)
    for cur_file in dir_list:
        path = os.path.join(file_dir,cur_file)
        if os.path.isfile(path):
            # print("{0} : is file!".format(cur_file))
            dir_files = os.path.join(file_dir, cur_file)
        if os.path.splitext(path)[1] == '.csv':
            csv_file = os.path.join(file_dir, cur_file)
            # print(os.path.join(file_dir, cur_file))
            # print(csv_file)
            list_csv.append(csv_file)
        if os.path.isdir(path):
            # print("{0} : is dir".format(cur_file))
            # print(os.path.join(file_dir, cur_file))
            list_dir(path)
    return list_csv
```